Synthèse d'Images
Travaux Pratiques

OpenGL Classique

L'objectif de ce TP est de développer un programme interactif en OpenGL standard (version 1.2). On partira du code du programme localisé dans le répertoire src, disponible dans l'archive du tp. On s'appuiera sur la documentation disponible sur le site www.opengl.org. Le sujet est écrit pour une implémentation en C++, mais une implémentation en C (structures et fonctions) est autorisée. On commencera par décompresser l'archive, se rendre dans le répertoire src et lancer la commande make –f Makefile.linux (remplacer linux par cygwin pour une compilation sous Windows/cygwin). Une fois le programme compilé, mon pourra le lancer à l'aide de la commande ./main. Pour l'instant le programme doit afficher un écran noir. La philosophie générale du programme est :

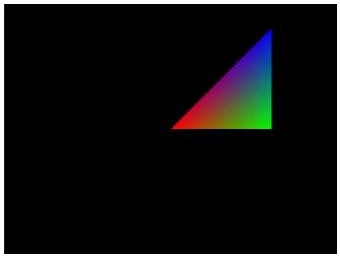
- définir et manipuler un ensemble de variable globale,
- les initialiser dans init ()
- les utiliser dans display ()

I. Rasterization de maillages et transformations

Observer le code du programme Main.cpp. Il s'agit d'une application interactive GLUT. La bibliothèque GLUT interface OpenGL avec votre système d'exploitation et son sous-système de fenêtrage.

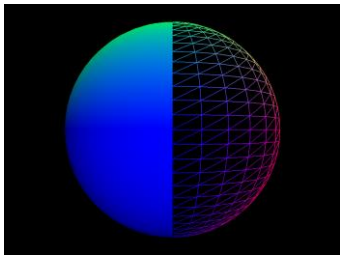
I.a On commencera par dessiner un triangle. Pour cela on placera le code suivant dans la fonction display () :

```
glBegin (GL_TRIANGLES);
glColor3f (1.0, 1.0, 1.0);
glVertex3f (0.0, 0.0, 0.0);
glVertex3f (1.0, 0.0, 0.0);
glVertex3f (1.0, 1.0, 0.0);
glEnd ();
```



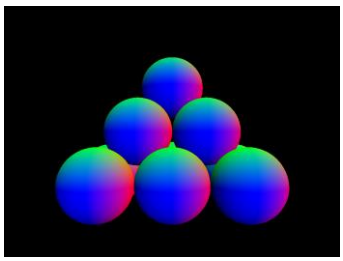
Ce code dessine un triangle blanc. On modifiera le code de manière à ce que chaque sommet ait une couleur différente. Pour cela on appellera glColor3f avec une valeur différente avant chaque émission de sommet (glVertex3f).

I.b Dessiner une sphère unitaire en dessinant un ensemble de triangles. On donnera à chaque sommet une couleur égale à sa position. On notera que la touche clavier w permet de passer du mode de remplissage des polygones pleins au mode filaire.



I.c Enfermer votre code de dessin dans une fonction glSphere (float x, float y, float z, float radius). Afin de placer les sphères et de leur donner une taille, on définira au sein de cette fonction la matrice de transformation du modèle. OpenGL fournit des commandes très simples pour manipuler la matrice courantes (un état). On insèrera notamment les lignes suivant dans glSphere :

```
glMatrixMode (GL_MODELVIEW); // Indique que l'on va désormais altérer la matrice modèle-vue
glPushMatrix (); // pousse la matrice courante sur un pile
```

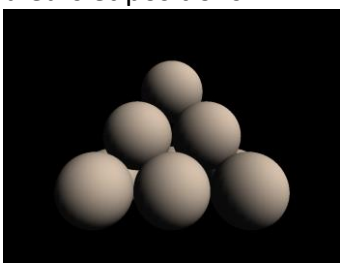


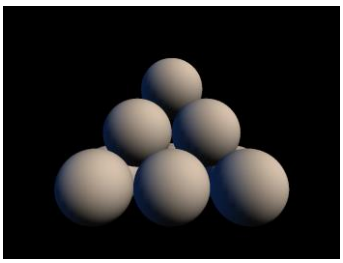
glTranslatef (x, y, z); // applique une translation à la matrice
[…] // dessin des polygones (glVertex3f, etc), dans le repère définit par la matrice model-vue
glPopMatrix (); // replace la matrice modèle vue courante original
On utilisera la fonction pour dessiner une pyramide de sphères.

II. Lumière, matériaux et textures en OpenGL (v1.2)

Dans cette partie, on active le modèle d'éclairage d'OpenGL en insérant glEnable (GL_LIGHTING) dans la fonction init(). A ce stade, il faut fournir une normale aux sommets de la surface. Comme pour la couleur, on changera l'état « normale » avant chaque émission de sommet (glVertex3f) à l'aide de la fonction glNormal3f (float nx, float ny, float nz). On modifiera glSphere en conséquence.

II.a Dans la fonction init (), activer 2 sources de lumières OpenGL, de couleurs et positions différentes (voir ici pour un détail des paramètres possible). Voici le code pour activer par exemple la source N° 0 :



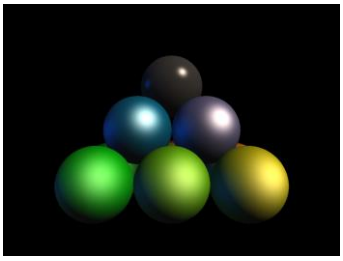


```
GLfloat light_position[4] = {10.0f, 10.0f, 10.0f, 1.0f};
GLfloat color[4] = {1.0f, 1.0f, 0.9f, 1.0f};
glLightfv (GL_LIGHT0, GL_POSITION, light_position);  // On place la source N° 0 en (10,10,10)
glLightfv (GL_LIGHT0, GL_DIFFUSE, color); // On lui donne légèrement orangée
glLightfv (GL_LIGHT0, GL_SPECULAR, color); // Une hérésie, mais OpenGL est conçu comme cela
glEnable (GL_LIGHT0); // On active la source 0
```

Il est possible de créer en standard jusqu'à 8 sources (GL_LIGHT0…8). Lier ces sources lumineuses à des touches clavier numériques (fonction keyboard ()) de manière à pouvoir les (des)activer individuellement (glEnable (GL_LIGHTX) / glDisable (GL_LIGHTX)). Pour reproduire l'exemple de droite, voici les propriétés des sources :

- source 1 : position = (10.0f, 10.0f, 10.0f), couleur = (1.0f, 0.9f, 0.8f),
- source 2 : position = (-10.0f, 0.0f, -1.0f), couleur = (0.0f, 0.1f, 0.3f),

II.b On souhaite maintenant spécifier un matériau particulier pour les objets de la scène. De la même manière que pour les autres paramètres, il s'agit un état courant à modifier dans OpenGL. On se propose d'associer un matériau à chaque, en créant une fonction de dessin enrichie :

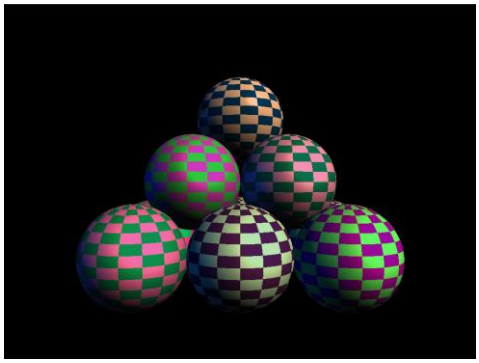


```
void glSphereWithMat (float x, float y, float z, float r,
                      float difR, float difG, float difB,
                      float specR, float specG, float specB,
                      float shininess)
```

Cette fonction mettra en place le matériau courant (couleur diffuse, couleur spéculaire et brillance) à l'aide des commandes OpenGL pour les matériaux :

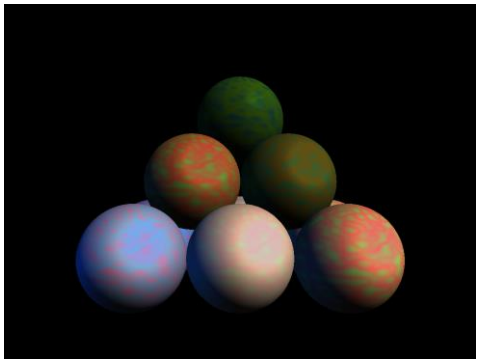
```
GLfloat material_color[4] = {difR, difG, difB, 1.0f};
GLfloat material_specular[4] = {specR, specG, specB,1.0};
glMaterialfv (GL_FRONT_AND_BACK, GL_SPECULAR, material_specular);
glMaterialfv (GL_FRONT_AND_BACK, GL_DIFFUSE, material_color);
glMaterialf (GL_FRONT_AND_BACK, GL_SHININESS, shininess);
```

II.c On se propose maintenant d'appliquer une texture aux objets. Pour cela, on commencera par équiper les surfaces d'une paramétrisation. Dans le cas d'une sphère, on pourra utiliser les coordonnées polaires des points en guise de paramétrisation. Pour cela, on va spécifier dans glSphere, pour chaque sommet, sa paire de coordonnée de texture à l'aide la commande glTexCoord2f (float u, float v) à appeler comme toujours avant l'emission du sommet (glVertex3f). La surface de nos sphère est maintenant prête à recevoir une texture couleur. Pour cela, il faut générer une texture. On créera un fonction void genCheckerboard (unsigned int width, unsigned int height, unsigned char * image) qui remplira le tableau image (préalablement alloué dans init (), ayant pour taille 3*width*height) avec un damier bleu et rouge (ou autre). On appelera cette fonction dans la fonction init () pour remplir une image. Ensuite, on construira une texture OpenGL à l'aide des commandes suivantes :



```
GLint texture; // Identifiant opengl de la texture
gEnable (GL_TEXTURE_2D); // Activation de la texturation 2D
glGenTextures (1, &texture); // Génération d'une texture OpenGL
glBindTexture (GL_TEXTURE_2D, texID); // Activation de la texture comme texture courante
// les 4 lignes suivantes paramètre le filtrage de texture ainsi que sa répétition au-delà du carré unitaire
glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
// La commande suivante remplit la texture (sur GPU) avec les données de l'image
glTexImage2D (GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, image);
```

Enfin, on utilisera cette texture lors du dessin de nos sphères. Pour cela, dans la fonction glSphereWithMat, juste avant d'appeler glSphere, on activera la texture à l'aide de la commande :
glBindTexture (GL_TEXTURE_2D, texture);

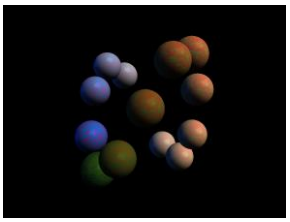
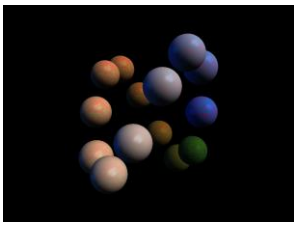


Bonus
A faire la fin.
Remplacer la fonction genCheckerBoardImage par une fonction genPerlinNoiseImage, remplissant l'image à l'aide du bruit de Perlin. L'archive tu TP contient également un ensemble d'images au format PPM dans le répertoire textures. Implémenter une fonction loadImage et utiliser les images chargées comme textures.

III. Animation et Interaction

Jusqu'à présent, notre application est statique. On se propose de lui ajouter animation et interaction.

III.a Le corps de la fonction idle () est nul pour l'instant. Cette fonction est exécutée en tâche de fond permanente. Insérer la commande glutPostRedisplay () dans idle afin d'obliger le programme à mettre à jour l'affichage aussi souvent que possible. Créer ensuite une variable globale static float currentTime. GLUT fournit une fonction pour récupérer le temps courant en milliseconds.

Placer dans idle () le code suivant :
currentTime = glutGet ((GLenum)GLUT_ELAPSED_TIME);
A partir de maintenant, la variable globale currentTime contient le temps courant. On animera une liste de sphères en les déplaçant le long d'une trajectoire fermée. Pour cela, on exprimera leur position (fonction glSphere) sur la trajectoire en fonction du temps.

III.b On souhaite maintenant contrôler interactivement la vitesse des sphères. Pour cela, créer une variable globale static float acceleration = 0.0 et utiliser ce facteur dans glSphere. Associer maintenant dans la fonction keyboard les touche + et – à l'augmentation ou la réduction de l'accélération. Il sera utile de mesurer le temps écoulé entre chaque mise à jour de l'affichage écran pour correctement contrôler la position des sphères à partir de la vitesse, et la vitesse à partir de l'accélération.

III.c Jusqu'à présente, la caméra a une position fixe. On se propose de la contrôler à l'aide de la souris et des fonctions mouse et motion. La fonction void mouse (int button, int state, int x, int y) est automatiquement appelée par GLUT à chaque évènement avec pour état state (GLUT_UP ou GLUT_DOWN) du bouton button (GLUT_LEFT_BUTTON par exemple) aux coordonnées x,y à l'écran. La fonction void motion (int x, int y) est appelée à chaque mouvement de la souris aux coordonnées x,y.
On mettra en place les variables globales nécessaires au controle de la position de la caméra, ainsi que la mise à jour la matrice modèle-vue à partir de ces variables et à l'aide de la fonction gluLookAt (voir l'utilisation faîte dans reshape). On pourra en particulier associer les coordonnées du pixel survolé par le pointeur souris aux coordonnées polaires de la caméra (camPosX/Y/Z) dans le repère de la cible. Rappel : avant d'employer gluLookAt, il faut repartir d'une matrice identité modèle-vue identité (glLoadIdentity (), qui s'applique à la matrice couramment activée, avec glMatrixMode (GL_MODELVIEW) pour activer la matrice modèle-vue par exemple).